The Making of Geneva: A Portable Document System

Max Rottenkolber <max@mr.gy>

Friday, 12 May 2017

Geneva (https://github.com/inters/geneva/#open-geneva) is a tool for authoring portable documents. Portable in the sense that they can be rendered to different formats, but also in the sense that the document structure is agnostic to any given serialization format. Geneva features a formally defined document model, an API for building documents, and a human readable, text based markup language. Also included are rendering back ends for HTML, LaTeX, and plain text. It is now available on *Quicklisp* (https://www.quicklisp.org/beta/).

Nothing Geneva does is new, it is predated by many solutions to the problems it solves such as *DocBook* (https://en.wikipedia.org/wiki/DocBook), *AsciiDoc* (https://en.wikipedia.org/wiki/AsciiDoc), and *Markdown* (https://en.wikipedia.org/wiki/Markdown). Heck, even within the Common Lisp community Geneva could not possibly be considered novel. Fernando Borretti wrote *CommonDoc* (https://github.com/CommonDoc) which does everything Geneva does and more, as far as I understand. Apparently he also did what took me close to five years in less than one.[1] *Sigh.* What follows is the tale of why and how it came to Geneva. I want to talk about the insights I gained from working on Geneva as well as the design decisions I made along the lines.

- 1. I admit that I was in no hurry.

Not Invented Here

The first *commit* (https://github.com/inters/geneva/commit/347279d9d93bb3dbf2cc523a8bc3f1b26cf4e762) dates back to December 2012, but I vividly remember writing and typesetting my *thesis* (https://github.com/eugeneia/microlisp) using an early prototype of Geneva in 2011. I can not accurately recall why exactly I decided to roll my own document toolchain. Its safe to assume that a fair amount of youthful stubbornness was involved. In some way, reinventing wheels is how I learned most of what I know today. And then again, I always had an excessive interest in computer assisted typesetting. If I could go back in time, would I invent Geneva again? No way. Nonetheless I do not regret the effort spent. Implementing Geneva gave me a thorough understanding of the problem at hand, and forced me to think about details that would otherwise not have crossed my mind.

On the bright side, I could not anticipate back then how important writing would become to me. Nowadays my obsession with technical documentation is a valuable professional skill. Creative writing has become an endeared spare-time activity of mine. Secretly, I want to become a journalist one day. Having a typesetting tool, that works exactly the way I envisioned it to, provides a lot of value to me. It goes without saying that I do not intend to persuade others to use Geneva.

A Simple Document Model

The semantics of a Geneva document are described in the *Geneva Document Specification* (http://inters.co/geneva/geneva-document.html). This specification is analogous to DocBook's XML schema. Unlike DocBook, a Geneva document is not defined in terms of an existing serialization format such as XML. The reasoning behind this decision is avoiding technical debt. Any Geneva implementation can choose how it represents documents in memory as long as it preserves their logical structure. Requirements for a specific character encoding are avoided for the same reason.

A formal specification of document semantics is important for both implementers and document authors alike. An implementer wants to know precisely what types of content he needs to deal with, and he needs to ensure that his program preserves document semantics. A document author needs to be aware of the semantics of a document in order to be able to express his intent. For instance, you will find that Markdown is poorly specified. Markdown's semantics could be considered the shared subset of its most popular implementations, but John Gruber might disagree.

Having defined a document model divides the remaining parts of the system into two categories:

- Front end programs that produce documents from input data
- Back end programs that produce rendered artifacts from documents

In terms of document features one could easily argue that Geneva is underpowered. It has a notion of *rich text*, which is a sequence of text that contain bold, italic, fixed-width (typewriter font), and hyperlink segments in addition to plain old text. Combinations of these are not supported. E.g., you can not have a text segment be both bold *and* italic. A document can contain paragraphs, listings, three types of figures, and sections. There is only one type of listing: the non-enumerated list. The supported types of figures are tables, verbatim text, and *generic media*. The latter is just a reference to an external resource, and it is the implementations responsibility to decide what to do with it. There is no concept of block quotation yet. Each figure has a *description* which is a paragraph "attached" to the figure. A section consists of a title and a sub-document.

This exercise in minimalism yields the tremendous advantage of simplicity. The front ends and back ends built around the document specification profit from its simplicity by being able to be simpler themselves. It also means there is less complexity for document authors to deal with, and therefore, simplifies what is the purpose of Geneva: writing. On the other hand the lack of features can be limiting for document authors. In practice this can mean substituting missing features with combinations of available features or avoiding certain typographic elements altogether—both can be unsatisfactory. Ultimately I learned to value simplicity over feature parity with existing typesetting systems. The process of adding features to Geneva has always been insanely conservative. For instance, labeled hyperlinks were a very late addition—it took me a long time to admit that I really need them.

Generally, I would prefer to add low level features, that can be combined to achieve typographic goals, over high level features. For instance: currently footnotes can be set using listings and manual enumeration. While this may seem unsatisfying, a higher level preprocessor could do this automatically.

A Lightweight Markup Language

Mk2 (http://inters.co/geneva/mk2.html) is a human readable and writable markup language—similar to Markdown—designed for authoring Geneva documents. I did think about simply implementing a Markdown front end for Geneva, but decided against it for two reasons. First, the syntax of Markdown is not formally defined. Implementing a parser for a language whose syntax is practically undefined is an impossible task. Obviously, there are a lot of people who wrote parsers for Markdown, but I like to argue that most, if not all, of these parsers are incorrect. Second, Markdown's semantics are defined in terms of HTML. That means Geneva would have to either support only a subset of Markdown, or introduce unnecessary impedance mismatch and technical debt. Starting a new markup language from scratch enabled me to gear it towards Geneva specifically. It also offered me a blank sheet of paper to draw on, and the opportunity to (hopefully) improve upon existing language designs.

Mk2 is formally defined by a context-free grammar.[1] The language is a precise representation of a Geneva document, meaning that any Geneva document can be serialized using Mk2.[2] The grammar is intended to be easy to read and write for humans, and its design is based on considerations of how humans interact with a markup language. The Mk2 parser will not ignore syntactic errors silently and attempt to "do the right thing"—being forgiving to the user is the responsibility of the Mk2 grammar.

When writing using a markup language, the most common error is to unintentionally invoke a language construct, because the user forgot to escape a *special token*. Thus, situations where escaping a special token is necessary should be as rare as possible. Derived from this premise are three design goals at the heart of Mk2:

- The number of special tokens should be as small as possible in order to minimize the number of tokens authors need to be aware of
- A special token should have no more than one meaning as to avoid ambiguity and complexity
- The context in which a special token bears meaning should be as small as possible

One aspect of Mk2 that might seem ill-conceived is the section syntax. In Mk2, sections start with the "<" token—followed by a title and the sections contents—and end with the ">" token. This mechanism allows sections to be nested, and their hierarchy to be derived from that nesting. Yet the need to close sections is the source for a common kind of error: failing to close a section. For instance, Markdown manages to avoid this problem by requiring its users to explicitly describe the section hierarchy using the header syntax. The upside of Mk2's more high level section syntax is that it does not require the author to think about explicit header levels. In practice this decision yields advantages when manually editing the section hierarchy of a document as well as for composing documents from multiple files.

- 1. The *string* axiom in used in Mk2's specification may lead you to believe otherwise, but it only serves conciseness and can be translated to a context-free grammar for each individual occurrence.
- 2. Geneva includes both front and back end programs for Mk2, see *Reading and Writing Mk2 Files* (http://inters.co/geneva/open-geneva.html#section-2).

Digital vs Print

Geneva comes with a HTML back end for the Web, and a LaTeX back end for print media. A Geneva document can be translated to HTML seamlessly. Individual document elements become semantic HTML elements, which can then be styled using CSS. Automatically typesetting documents for print media is not as easy. Aside from physical limitations such as paper geometry, centuries of typesetting craft have set a high standard regarding the sophistication of print media. In an attempt to reach a middle ground between aesthetics and automation, I choose TeX as a compilation target. This choice means that Geneva can benefit from numerous high quality TeX implementations, and is able to reuse the tremendous amount of work that went into LaTeX over the years.

TeX manages to automate some aspects of sophisticated typesetting such as justification, spacing, and hyphenation, but fails to automate others such as table layout. Strictly speaking, TeX is still a manual typesetting system—it requires the user to set the document on a word by word basis. Even the aspects TeX does automate need to be bypassed in some cases where its algorithms produce unsatisfying results. Geneva, on the other hand, attempts to be a fully automatic typesetting system—its users should not be bothered beyond authoring document structure and content.

Geneva's
LaTeX back end (http://inters.co/geneva/open-geneva-api.html#section-6) reflects an uphill battle between two detrimental forces: automation and sophistication. It patches over many edge cases that are not properly handled by LaTeX while attempting to make as much use as possible of the features provided by it. The LaTeX back end accepts a "preamble" argument—somewhat analogous to a CSS stylesheet—allowing to insert

arbitrary (La)TeX code into the output. Depending on the LaTeX implementation used to process the resulting TeX files, a preamble might be required to enable Unicode support, if the input document requires it. For non-English documents, at the very least, a preamble that loads the LaTeX's "babel" package is required for hyphenation to work correctly.[1]

Ultimately, I think that TeX is not the right target for Geneva's print media back end. Achieving fully automatic and yet sophisticated typesetting is a very hard problem, and (for me personally) it is not worth solving. A crude and simple system such as HTML provides for everything needed to effectively digest written information. Ideally, Geneva would have an equally simple PDF back end, that sacrifices the fancy aesthetics of TeX for consistent and predictable output. Being enthusiastic about high quality typesetting, this realization was quite shocking to me at first. In the end I had to admit to myself, that in a digital world the effort required to do sophisticated typesetting outweighs its virtues. For now, Geneva's LaTeX back end produces reasonably satisfying results. For the future, I would like to add a simpler alternative based on *cl-pdf* (https://github.com/mbattyani/cl-pdf).

- 1. Hyphenation is language dependent, so in order to do it correctly it is necessary to know what language the document is written in.